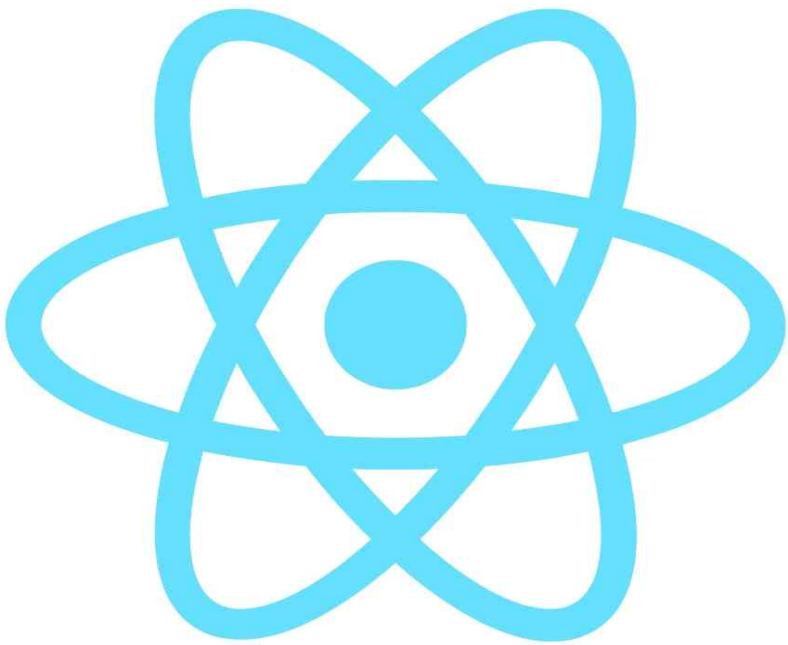
전공분야 보수교육

[정보기술개발]

프로젝트 기반의 리액트 기초



React

한국기술교육대학교
능력개발교육원

인천일보아카데미

프로젝트 기반의 리액트 기초

발행 2023년 10월 4일

지은이 황인철

발행인 박신석

발행처 인천일보 아카데미

주소 인천시 미추홀구 매소홀로 488번길 6-32, 태승빌딩 5층

전화 032-867-3332

전자우편 inchoriya@icia.co.kr

홈페이지 www.icia.co.kr

프로젝트 기반의 리액트 기초

전공분야 보수교육

황인철 지음

한국기술교육대학교 능력개발교육원

인천일보아카데미

연수과정 개요

1. 과정명 : [정보기술개발] 프로젝트 기반의 리액트 기초

2. 교육기간(시간) : 2일(12시간)

 - 2023년 10월 14일(토) - 15일(일) : 집체 수업

3. 교육대상자 : 정보통신 분야 직업훈련교·강사, 직업훈련종사자 등

4. 교육수준 : 중급

5. 교육형태 : 집체(이론 30%, 실습 70%)

6. 선수능력(과정) : HTML, CSS, JavaScript 기초지식

   - 선수과정 : 프로젝트 기반의 프론트엔드 프로그래밍(권장)

7. 교육장소 : **인천일보아카데미**
   (인천광역시 미추홀구 매소홀로488번길 6-32)

8. 교육목표 : 리액트의 기본 문법을 익히고 게시판 만들기 실습을 통해 리액트의 응용 문법을 학습한다.

연수과정 편성 총괄표

<table>
<tr><td rowspan="3">관련NCS</td><td>NCS능력단위</td><td>NCS능력단위요소</td></tr>
<tr><td>. 화면 구현
(2001020225_19v5)</td><td>. UI 설계 확인하기(2001020225_19v5.1)
. UI 구현하기(2001020225_19v5.2)</td></tr>
<tr><td>. SQL활용
(2001020413_19v4)</td><td>. 기본 SQL 작성하기(2001020413_19v4.1)
. 고급 SQL 작성하기(2001020413_19v4.2)</td></tr>
<tr><td colspan="2">20.정보통신 > 01.정보기술> 02.정보기술개발> 02.응용SW엔지니어링</td></tr>
<tr><td>교수-학습
방법</td><td colspan="2">. 이론 강의 및 실습
. 교구를 활용한 실습 등</td></tr>
<tr><td>평가방법</td><td colspan="2">. 수행평가(실습과제 제출)</td></tr>
<tr><td>활용장비</td><td colspan="2">. react.js, node.js, Visual Studio Code</td></tr>
<tr><td>활용교재</td><td colspan="2">. 자체유인물</td></tr>
</table>

일차
01

리액트 시작하기

STEP 01 | 리액트 시작하기

리액트 소개 및 개요
리액트 개발환경 구축

STEP 02 | 리액트 프로젝트 분석하기

리액트 프로젝트 생성
리액트 프로젝트 구조 분석

STEP 03 | 화면 전환하기

리액트 router 및 component
리액트 커뮤니티 프로젝트 실습
(component 작성, 기능 소개, api 설계)

STEP 04 | form 다루기

form, useState(), hook 다루기
리액트 커뮤니티 프로젝트 실습
(게시글 작성을 위한 form)

STEP 05 | 데이터 베이스 연동하기 I

mysql 데이터베이스 개발환경 구축
리액트 커뮤니티 프로젝트 실습
(프로젝트를 위한 데이터베이스 실습)

STEP 06 | 데이터 베이스 연동하기II

데이터베이스 연동환경 구축
리액트 커뮤니티 프로젝트 실습
(프로젝트를 위한 CRUD 쿼리문)

일차
02

리액트 커뮤니티 프로젝트

STEP 07 | form 데이터 저장하기
요청된 데이터 DB 저장하기
리액트 커뮤니티 프로젝트 실습
(게시글 작성을 위한 함수 개발)

STEP 08 | 리액트 커뮤니티 프로젝트
게시글 목록 기능 구현하기 I
client 에서 게시글 목록 데이터 요청하기
조회데이터 리액트 component 로 가져오기

STEP 09 | 부트스트랩 문법
게시글 목록 기능 구현하기II
목록 데이터 화면에 출력하기
부트스트랩 배지, HTML 속성 초기화

STEP 10 | 리액트 커뮤니티 프로젝트 I
게시글 조회 기능 구현하기 I
useParams() hook 다루기, 데이터 요청하기

STEP 11 | 리액트 커뮤니티 프로젝트II
게시글 조회 기능 구현하기II
데이터 화면에 요청 및 출력하기

STEP 12 | 리액트 커뮤니티 프로젝트III
게시글 삭제 기능 구현하기 및 마무리
게시글 삭제 요청하기

1일차

리액트 시작하기

STEP 01

리액트 준비하기

리액트를 배우기 위해 필요한 기초 지식에 대해서 학습한다.
웹페이지를 구성하는 HTML 과 웹페이지의 동적인 부분을 구현하는 자바스크립트 문법과
자바스크립트 표준 규격인 ECMAScript 문법에 대해서도 알아본다.

❑ HTML(Hyper Text Markup Language)

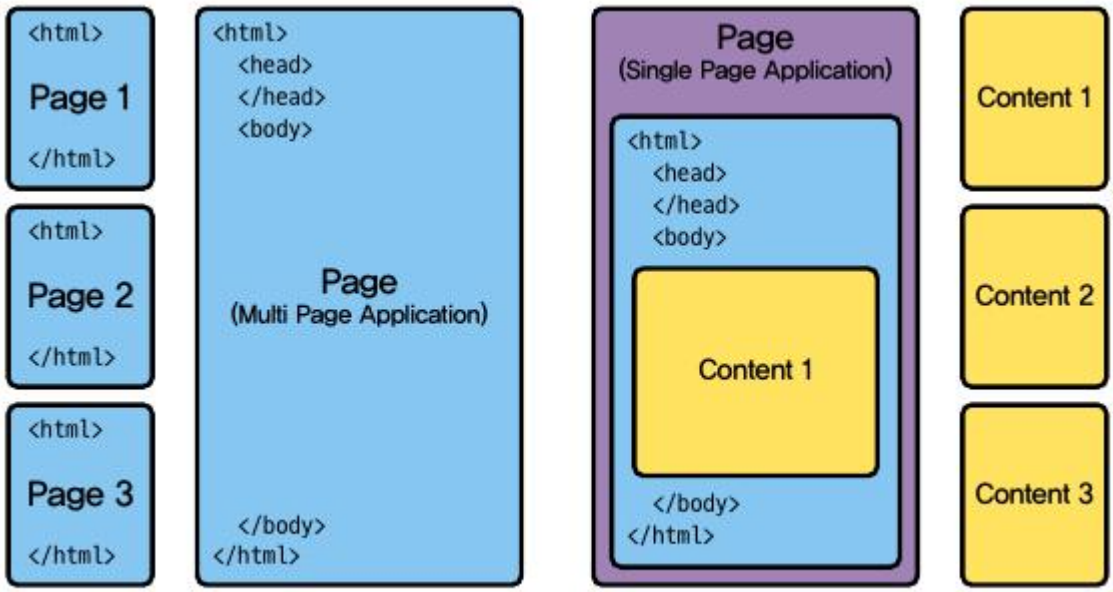
- 웹사이트의 뼈대를 구성하는 태그

```
<html>
    <head>
    </head>
    <body>
    </body>
</html>
```

〈html〉 태그 : 말 그대로 HTML 의 시작과 끝을 알리는 태그
〈head〉 태그 : 이 웹사이트가 어떤 웹사이트인지 알 수 있는 여러 가지 속성(제목, 설명 등)을 담고 있다.
〈body〉 태그 : 실제 웹사이트에서 보이는 콘텐츠, 실제로 웹브라우저에서 보게 되는 내용

❑ SPA(Single Page Application)

- 복잡한 사이트에 접속하면 여러 가지 버튼이나 탭을 누르면서 여러 페이지를 이동하게 되는데 그때마다 브라우저에 나오는 내용이 달라지는 볼 수 있다. 내용이 바뀐다는 것은 각 페이지별로 HTML 이 존재하며, 페이지를 이동하게 될 경우 브라우저에서 해당 페이지의 HTML 파일을 받아와서 화면에 표시해 준다. 이와 같이 수많은 HTML 파일을 관리하기 힘든 문제를 해결하기 위해 나오게 된 것이 SPA 이다. SPA 는 말 그대로 하나의 페이지만 존재하는 웹사이트다.



▶ MPA vs. SPA

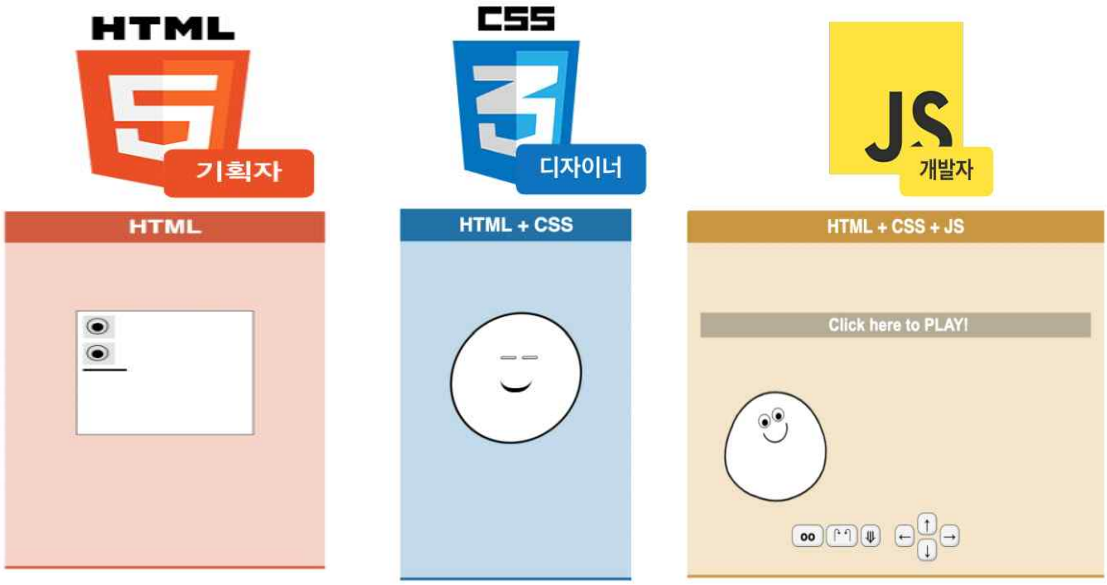
왼쪽에 보이는 것이 전통적인 웹 애플리케이션 방식이고, 오른쪽에 보이는 것이 SPA 이다.
HTML 파일 대신 해당 콘텐츠를 가져와 동적으로 <body> 태그 내부를 채워 넣는 방식이다.

❑ CSS(Cascading Style Sheets)

- HTML 로 웹사이트의 구조를 만들었다면, 그 위에 CSS 를 사용하여 디자인을 입히는 형태

❑ JS(JavaScript)

- 자바스크립트의 정식 명칭은 ECMAScript 이다. ECMAScript 의 글자를 따서 ES5, ES6 과 같이 표현하다. 2022 년 기준 최신버전은 ES13 이고, 본 강의에서는 ES6 를 다루도록 한다. HTM 이 웹사이트의 뼈대를 구성하는 역할을 한다면, 자바스크립트는 웹사이트가 살아 움직이도록 생명을 불어 넣는 역할을 한다.



❑ ES6 문법

- ES 란, ECMAScript 의 약자이며 자바스크립트의 표준, 규격을 나타내는 용어

- **let, const 키워드**
  - let : 재선언 불가, 재할당 가능
  - const : 상수 선언 키워드
  - 기존 var 키워드만 있었을 때보다 예측 가능한 코드를 작성할 수 있게 되었다.

- **템플릿 리터럴**
  - 사용법은 ` ` (back tick)으로 가능하다.
  - ${ } 중괄호 앞에 달러 표시를 통해 자바스크립트 표현식 사용이 가능하다.

```
// ES5
var str1 = ', ';
var str2 = 'World!';
var str3 = 'Hello' + str1 + str2;

// ES6
let str1 = ', ';
let str2 = 'World!';
let str3 = `Hello ${str1} ${str2}`;
```

- **화살표 함수 : 우측의 표현식을 평가하고 평과 결과를 반환**

```
// ES5 함수
function myFunc(name){
        return "이름 : " + name;
}

// ES6 화살표 함수
const myFunc = (name) => {
        return `이름 : ${name}`;
}

// return 키워드 생략가능
const myFunc = (name) => `이름 : ${name}`;

console.log(myFunc('황인철'));
// 출력결과 => 이름 : 황인철
```

- **구조 분해 할당**
  - '펼치다' 란 뜻으로 객체나 배열에서 사용하며, **값을 해체**한 후 개별 **값을 변수에 새로 할당**하는 과정

```
const student ={
        name : "황인철",
    age : 30
}

const { name, age } = student;
console.log(`name : ${name}, age : ${age}`);
// 출력 결과 => name : 황인철, age : 30
```

- **Multi-line String**
  - 문자열이 라인을 넘어가게 되면 '₩n' 과 덧셈 연산자를 사용했어야 했다.
  - 백틱을 사용하게 되면서 여러줄의 문자열 관리도 편해졌다.

```
// ES5
var str = '안녕하세요.\n' +
'보수교육에 오신 여러분을 환영합니다.\n'

// ES6
let str = `안녕하세요.
보수교육에 오신 여러분을 환영합니다.`;
```

STEP 02

리액트 시작하기

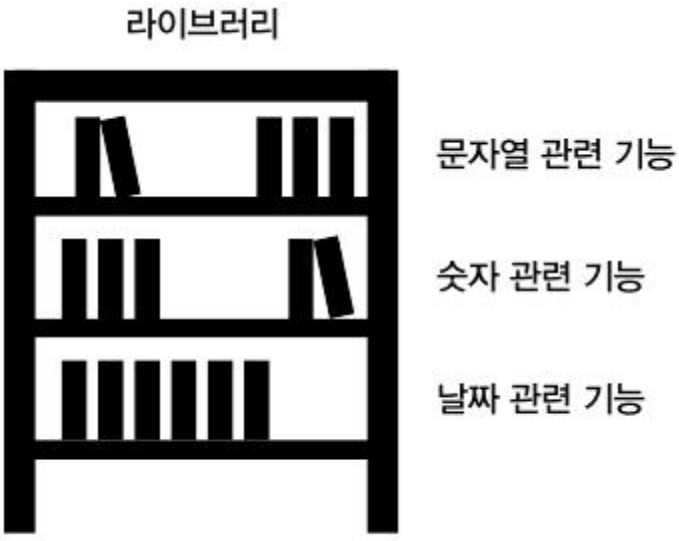
리액트에 대해서 간단히 알아보도록 한다. 어떤 라이브러리나 기술에 대해 배울 때는 먼저 그것의 정의에 대해 잘 알고 배우는 것이 좋다. 내가 배우고 있는 것이 무엇인지도 제대로 이해하지 못한다면 마치 바닷가의 모래 위에 성을 쌓는 것처럼 무의미해진다. 그런 의미에서 리액트의 정의와 개념에 대해 알아보도록 한다

❑ 리액트의 정의

- 리액트의 공식 웹사이트에서는 다음과 같이 리액트를 정의한다.

  A JavaScript library for building user interfaces.

보통 영단어 library는 도서관 이라는 의미로 사용되는데, 프로그래밍 언어에서는 자주 사용하는 기능을 정리해 모아 놓은 것이라는 의미를 가지고 있다.



▶ 프로그래밍 언어에서의 라이브러리를 책장으로 표현한 그림

사용자 인터페이스(User Interface)를 만들기 위한 기능 모음집을 UI 라이브러리라고 부르고, 리액트는 대표적인 자바스크립트 UI 라이브러리라고 할 수 있다.

❑ 리액트 개념 정리

- 리액트는 사용자와 웹사이트의 상호작용을 돕는 인터페이스를 만들기 위한 자바스크립트 모음집
- 사이트의 규모가 커질수록 수백 개의 페이지가 존재하는데 이것을 각각 페이지마다 HTML 로 만드는 것이 아니라 하나의 HTML 틀을 만들어 놓고, 사용자가 특정 페이지를 요청할 때 그 안에 해당 페이지의 내용을 채워서 보내주는 SPA(Single Page Application)을 빠르게 만들 수 있도록 해주는 도구

❑ 리액트의 장점

- 빠른 업데이트와 렌더링 속도

내부적으로 Virtual DOM(가상의 DOM)을 사용해서 업데이트해야 할 최소한의 부분을 검색해서 검색된 부분만 업데이트하고 다시 렌더링하면서 변경된 내용을 빠르게 사용자에게 보여준다.

- 컴포넌트 기반 구조

컴포넌트는 구성요소라는 뜻을 갖고 있는데 리액트에는 모든 페이지가 컴포넌트로 구성되어 있고, 하나의 컴포넌트는 또 다른 여러 개의 컴포넌트의 조합으로 구성될 수 있다. 마치 작은 레고 블록들이 모여서 하나의 완성된 모형이 되는 것과 비슷하다.

- 재사용성

객체지향 프로그래밍에서 등장하는 개념으로 다시 사용이 가능한 성질을 말한다. 재사용성이 높아지면 소프트웨어 개발 기간이 단축되고, 유지 보수가 용이하다. 또한 여러 모듈 간의 의존성이 낮아서 각 부분들이 잘 분리되고 쉽게 버그를 찾아서 수정할 수 있다.

- 메타(페이스북)이라는 지원군

세계 최대 IT 기업 중 하나인 메타에서 시작한 프로젝트로 꾸준히 버전 업데이트가 이뤄지며 발전하고 있다. 현재 시점에서 가장 인기 있는 라이브러리이며 최소 몇 년 동안은 그 영향력이 지속될 것이다. 메타에서 프로젝트를 종료하지 않는 한 계속 발전해 나갈 것으로 예상된다.

- 활발한 지식 공유 & 커뮤니티

현재의 영향력을 가지게 된 이유 중 하나로 개발 생태계와 커뮤니티를 꼽을 수 있다. 오픈소스 플랫폼인 깃허브의 리액트 프로젝트의 경우 모든 오픈소스를 통틀어 최상위급 수치를 보이고 있다. 또한 특정 기술의 생태계 규모를 판단하는 지표로 스택오보플로 웹사이트의 해당 기술 언급양을 보더라도 높은 수치를 보여준다.

- 모바일 앱 개발가능

리액트를 배운 이후에 리액트 네이티브라는 모바일 환경 UI 프레임워크를 사용하여 모바일 앱도 개발할 수 있다. iOS 는 스위프트(Swift)로, 안드로이드의 경우 코틀린(Kotlin)이라는 프로그래밍 언어를 배워야 하지만 리액트 네이티브를 사용하면 안드로이드앱과 iOS 앱을 동시에 출시할 수 있다.

❑ 리액트의 단점

- 방대한 학습량

리액트의 경우 다른 UI 라이브러리에 비해 학습할 내용이 많다. 본 강의에서는 리액트의 필수적이고 중요한 부분만 추려서 배우게 된다. 개발자마다 사용하는 방식이 다르고 계속해서 업데이트 되기 때문에 꾸준히 학습하고 내용을 숙지해야 실무 환경에서 원활하게 작업할 수 있다.

- 높은 상태 관리 복잡도

리액트에는 state 라는 굉장히 중요한 개념이 있다. state 는 쉽게 말해 리액트 컴포넌트의 상태를 의미하는데 업데이트를 할 때 바뀐 부분만 찾는다고 했는데 여기서 바뀐 부분이란 state 가 바뀐 컴포넌트를 의미한다. 성능 최적화를 위해 state 를 잘 관리하는 것이 중요하고, 규모가 커질수록 컴포넌트의 개수도 많아지면서 상태 관리의 복잡도도 증가한다.

❑ 개발환경 구축

• Node.js 와 npm 설치

Node.js 는 자바스크립트로 네트워크 애플리케이션을 개발할 수 있게 해주는 환경이다.

npm 은 Node.js 를 위한 패키지 매니저로 node package manager 의 약자이다.

(1) Node.js 홈페이지(https://nodejs.org/ko/)에서 설치 파일 다운로드해서 설치

(2) cmd 창 (win 키 + R 키 → cmd 입력 혹은 명령 프롬프트창)

node --version 을 입력해서 Node.js 의 버전 확인

npm --version 을 입력해서 npm 버전 확인(npm 은 Node.js 설치 시 함께 설치됨)

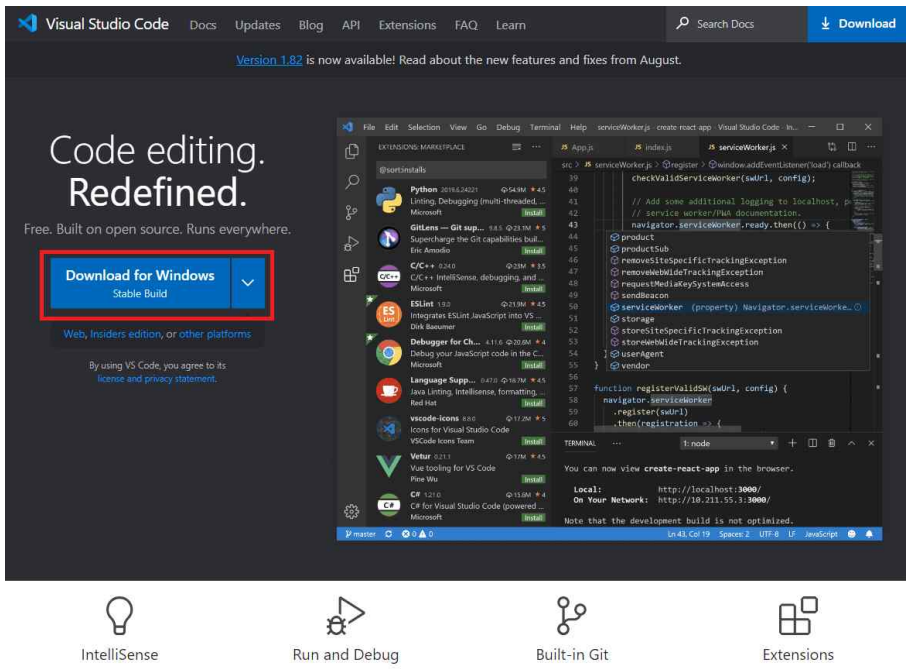
```
C:\Users\user>node --version
v18.18.0

C:\Users\user>npm --version
9.8.1
```

- VSCode 설치

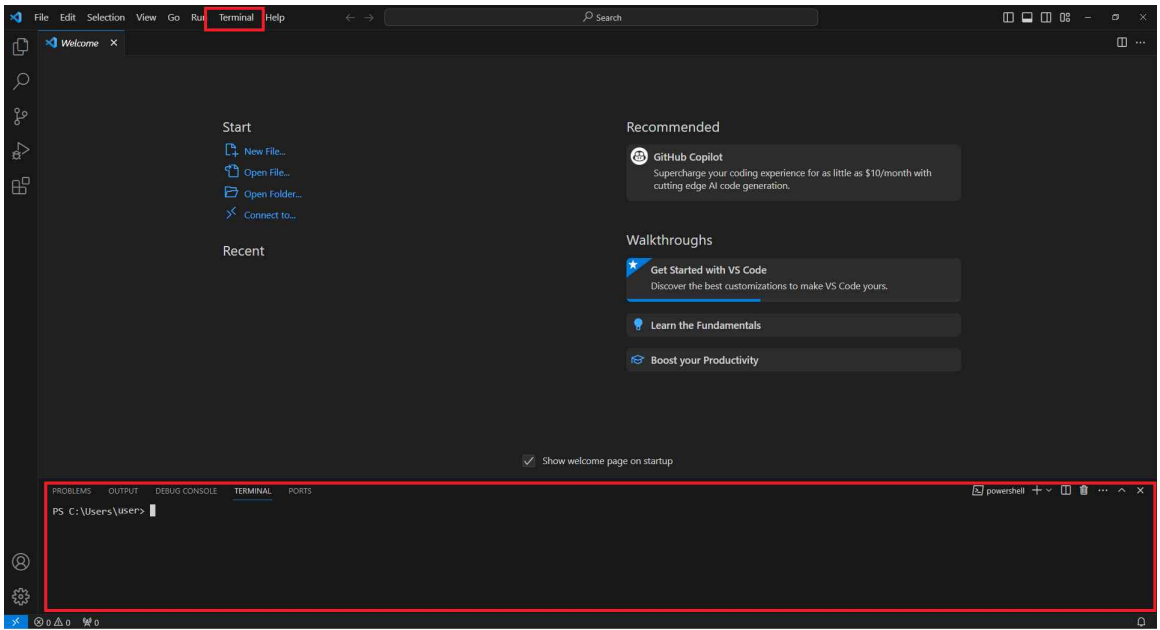
메모장 같은 텍스트 편집 프로그램을 사용해도 코딩을 할 수 있지만 코드 자동 정렬, 함수 참조 찾기 등의 부가적인 기능을 제공하는 프로그램을 사용하면 더 수월하게 코딩할 수 있다. 이러한 기능을 제공하는 프로그램을 IDE(Integrated Development Environment) 통합 개발 환경이라 한다. VSCode 는 리액트 개발에 적합한 IDE 로 본 강의에서 사용하도록 한다.

(1) VSCode 홈페이지(https://code.visualstudio.com/)에서 설치 파일 다운로드해서 설치



(2) 상단 메뉴의 Terminal > New Terminal 을 선택하거나 단축키 [Ctrl] + [Shift] + [`] 키

이제부터는 이 터미널을 이용해서 커맨드 라인 명령어를 실행한다.



❑ 프로젝트 생성하기

- create-react-app

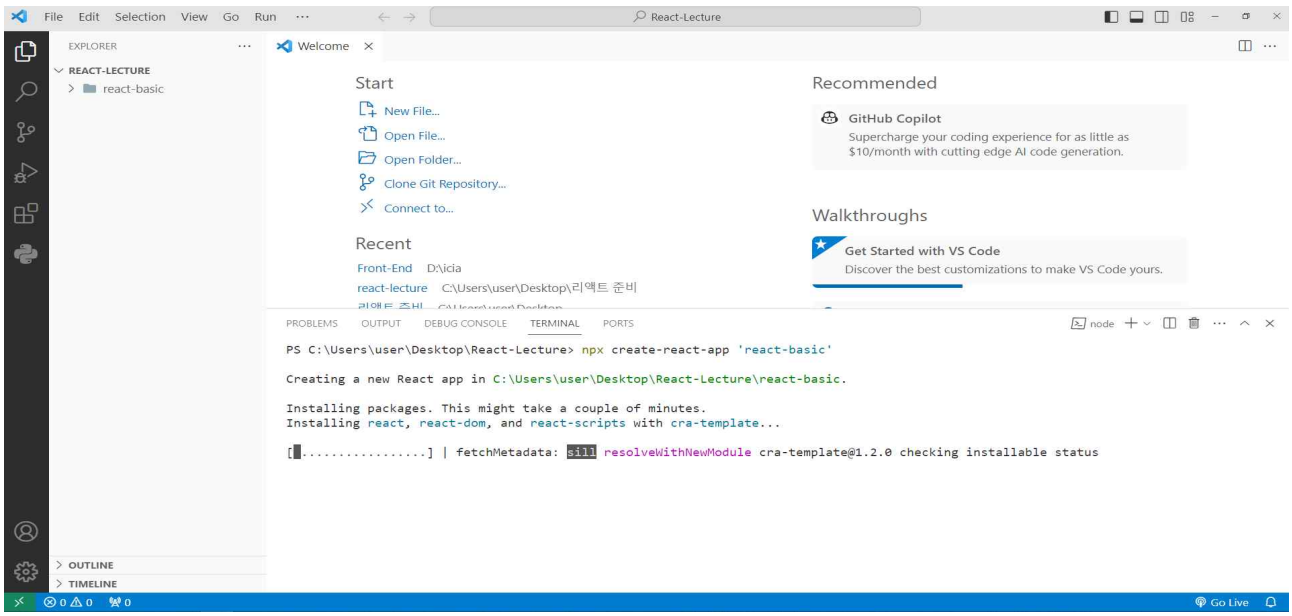
리액트를 개발할 때 주로 create-react-app 을 사용해서 프로젝트를 생성한다. 그러면 리액트로 웹 애플리케이션을 개발하는 데 필요한 모든 설정이 되어 있는 상태의 프로젝트를 생성해 준다. VSCode 의 상단 메뉴 File - Open Folder 를 선택해서 프로젝트를 생성할 폴더를 지정한다.

사용법)

```
npx create-react-app <project-name>
```

실제 사용 예제)

npx create-react-app ‘react-basic’ 입력



- create-react-app 'react-basic' 실행 오류시

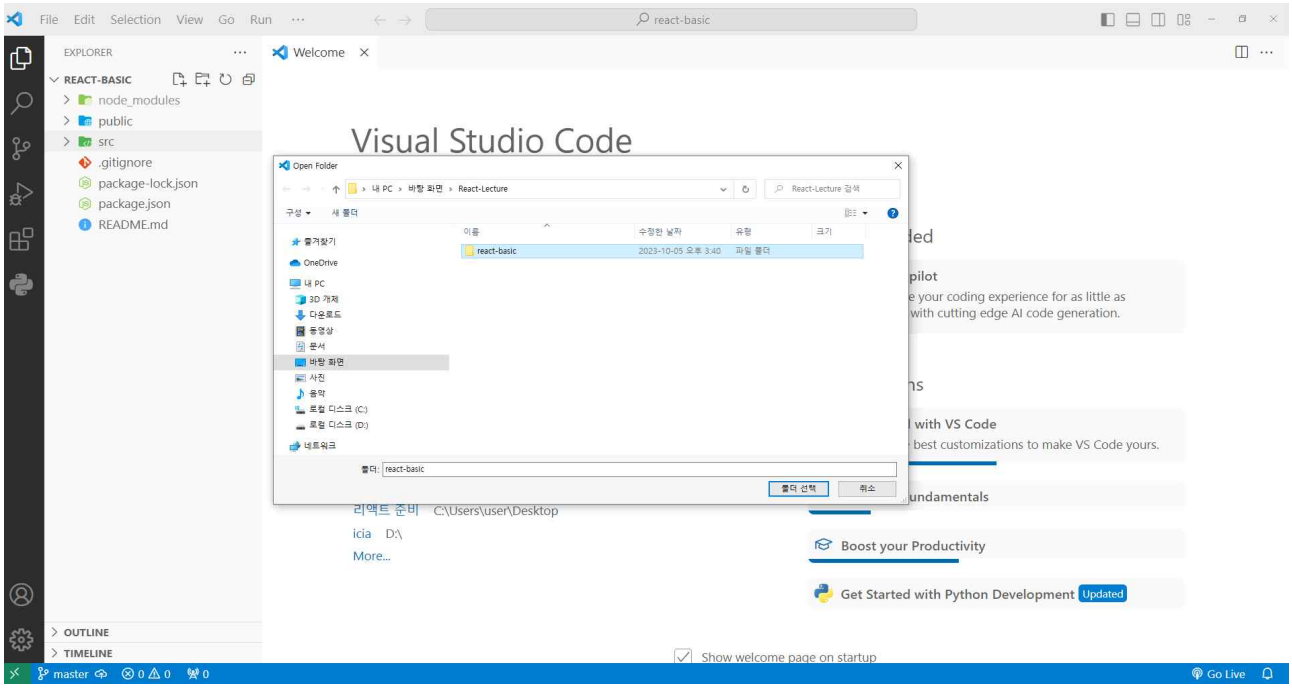
(1) npm uninstall -g create-react-app 실행

(2) npm install -g create-react-app 실행

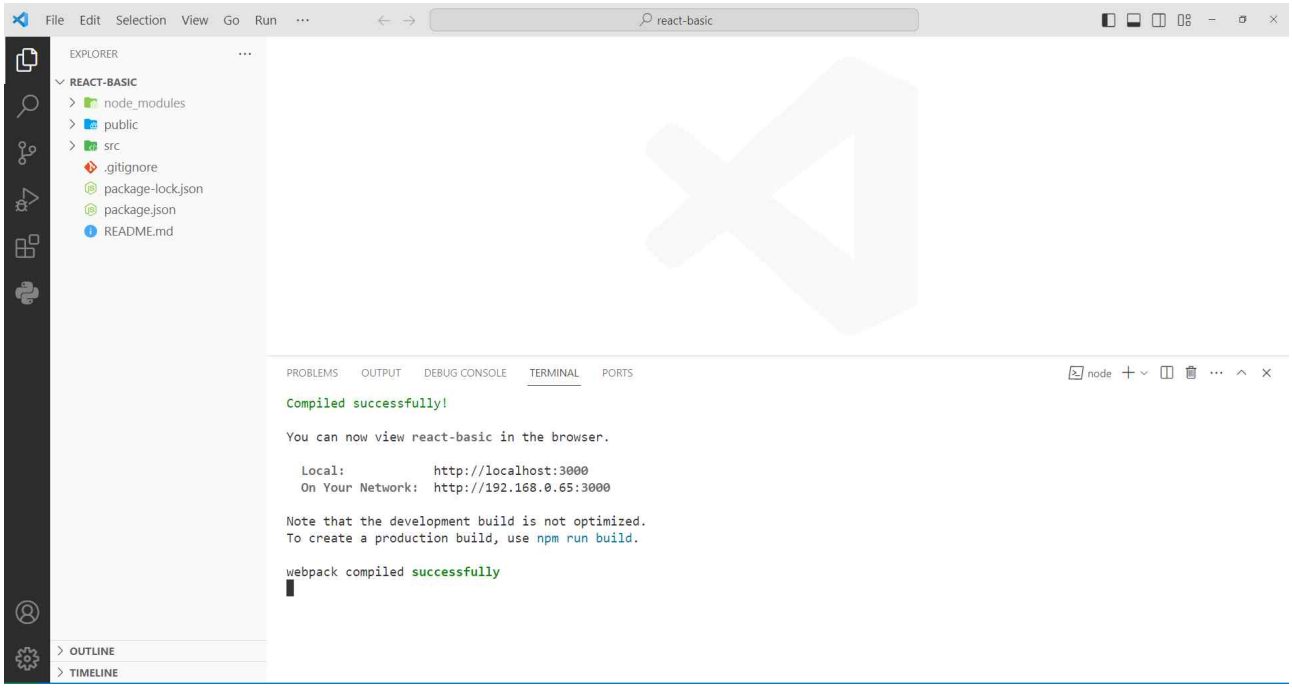
(3) 다시 npx create-react-app 'react-basic' 실행

- 프로젝트 실행

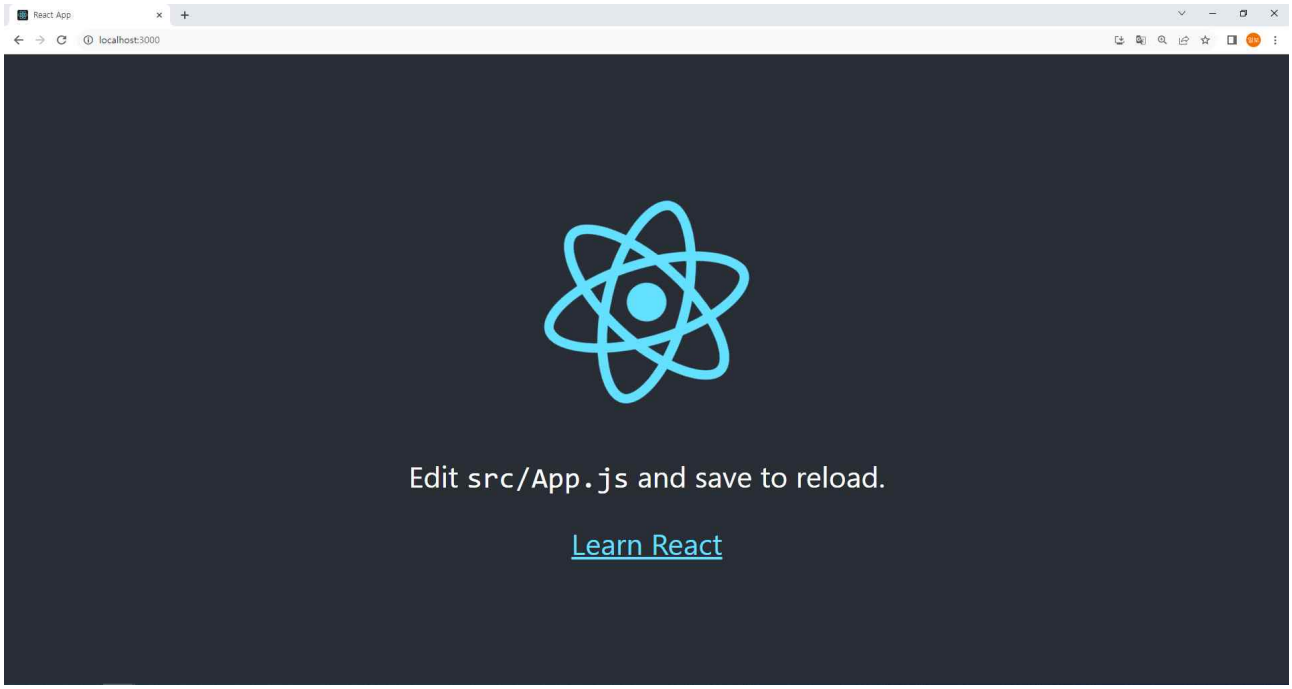
(1) VSCode의 상단 메뉴 File - Open Folder를 선택해서 생성된 프로젝트 폴더를 지정한다.



(2) npm start 입력



(3) 명령어 실행하고 조금 기다리면 자동으로 브라우저가 열리면서 http://localhost:3000 으로 접속



STEP 03

JSX 소개

JSX 는 리액트를 사용하여 개발할 때 거의 필수적으로 사용해야 한다. 그렇기 때문에 JSX 의 개념을 꼭 확실하게 이해하고 넘어가야 한다.

❑ JSX 란

- JSX 는 A syntax extension to JavaScript (자바스크립트의 확장 문법) 이다.
  JavaScript and XML.을 JSX 라고 부른다.

```
const element = <h1>Hello, world!</h1>
```

위 코드와 같이 const element 와 같은 자바스크립트 코드와 <h1>태그를 사용한 HTML 코드가 결합 되어 있는 코드를 JSX 코드라고 보면 된다.

❑ JSX 의 장점

- 코드가 간결해진다.

```
// JSX 사용
<div>Hello, {name}</div>

// JSX 사용 안함
React.createElement('div', null, `Hello, ${name}`);
```

JSX 를 사용한 코드의 경우, HTML 의 태그를 사용해서 name 이라는 변수가 들어간 인사말을 표현했고 JSX 를 사용하지 않은 코드의 경우 리액트의 createElement() 함수를 사용하여 동일한 작업을 수행하였다.

- 가독성이 향상된다.
  JSX 에 대해 모르는 사람이 보더라도 코드의 의미를 훨씬 빠르게 이해할 수 있고 가독성이 향상될수록 버그 또한 쉽게 발견할 수 있다는 장점이 있다.

❑ JSX 의 사용법

- JSX 는 자바스크립트 문법을 확장시킨 것이기 때문에, 모든 자바스크립트 문법을 지원한다. 여기에 추가로 XML 과 HTML 을 섞어서 사용하면 된다.

```
const name = 'icia';
const element = <h1>안녕, {name}</h1>

ReactDOM.render(
        element,
    document.getElementById('root');
);
```

XML 이나 HTML 코드를 사용하다가 중간에 자바스크립트 코드를 사용하고 싶으면 {name}과 같이 중괄호를 묶어서 사용해주면 된다.

❑ JSX 코드 작성해보기

- Student.jsx 라는 이름의 파일을 생성하고, 이 컴포넌트는 props 로 name 과 age 를 받아서 이를 출력하는 컴포넌트이다. JSX 를 모르는 사람이 봐도 어떤 역할을 하는지 대강 이해할 수 있는 수준이다. JSX 를 사용하면 이처럼 가독성이 높고 직관적인 코드를 작성할 수 있다.

```
import React from "react";

function Student(props){
    return(
        <div>
            <h1>{`이 학생의 이름은 ${props.name}입니다.`}</h1>
            <h2>{`이 학생은 ${props.age}살 입니다.`}</h2>
        </div>
    );
}

export default Student;
```

- 이번에는 Student 컴포넌트를 사용하는 상위 컴포넌트를 만들어 보도록 한다. 같은 폴더에 Students.jsx 파일을 새로 만들고 아래와 같이 코드를 작성한다.

```
import React from "react";
import Student from "./Student";

function Students(props){
    return(
        <div>
            <Student name="유재석" age={49} />
            <Student name="손흥민" age={32} />
            <Student name="아이유" age={28} />
        </div>
    );
}

export default Students;
```

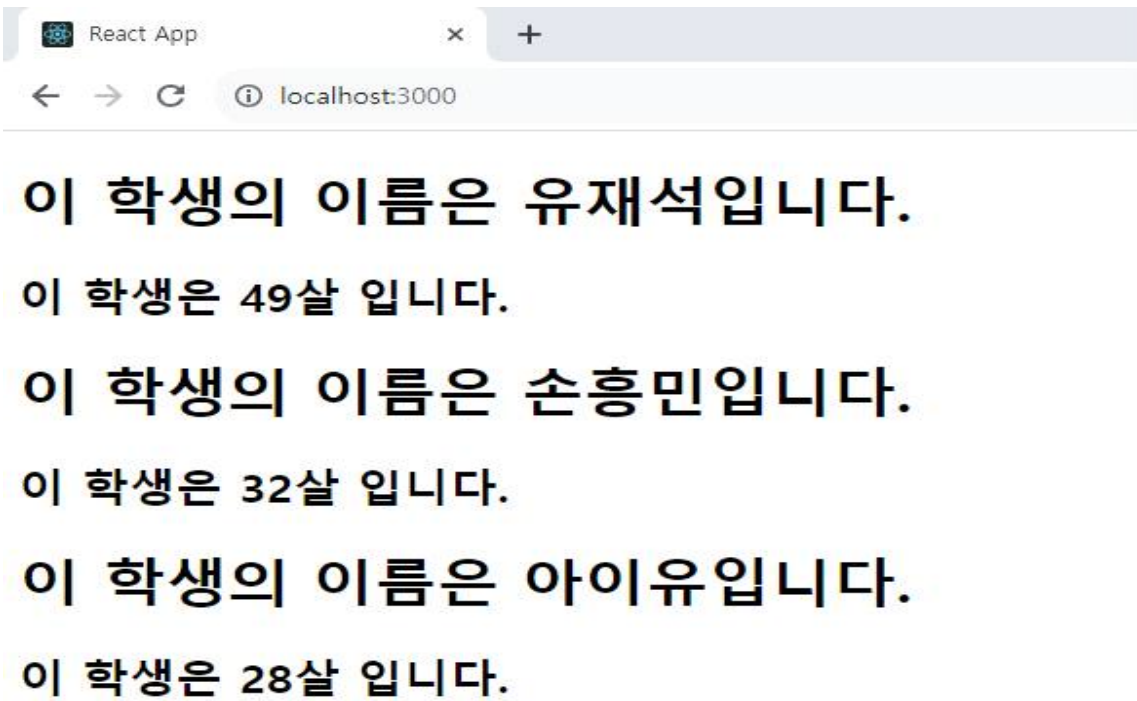
type 에 HTML 태그 이름이 문자열로 들어가는 경우, 엘리먼트는 해당 태그 이름을 가진 DOM Node 를 나타내고

- 우리가 만든 컴포넌트를 실제로 화면에 렌더링 하기 위해서 index.js 파일을 수정해야 한다.

```
import React from 'react';
import ReactDOM from 'react-dom';
import Students from './Students';


ReactDOM.render(
  <React.StrictMode>
    <Students />
  </React.StrictMode>,
  document.getElementById('root')
);
```

모든 작성이 끝난 후 VSCode 의 상단 메뉴 Terminal > New Terminal 을 눌러서 새로운 터미널을 하나 실행 시키고 npm start 명령어를 실행한다. 그러면 잠시 뒤 웹브라우저의 새창이 열리면서 http://localhost:3000 에 접속되면서 아래 그림처럼 작성한 내용대로 컴포넌트들이 렌더링 된 것을 확인 할 수 있다.



STEP 04

엘리먼트 렌더링

이 장에서는 리액트의 엘리먼트(element)라는 개념에 대해 배워본다. 앞서 리액트의 createElement() 함수를 사용해봤다. createElement() 함수는 이름 그대로 엘리먼트를 생성해주는 함수이다. 그렇다면 엘리먼트가 무엇이고 어떤 역할을 하는지, 엘리먼트가 렌더링 되는 과정에 대해서 자세히 알아보도록 한다.

❑ 엘리먼트 정의

- Elements are the smallest building blocks of React apps. 엘리먼트는 리액트 앱의 가작 장은 빌딩 블록들 이라는 의미가 된다. 엘리먼트는 원래 웹사이트에 대한 모든 정보를 담고 있는 객체인 DOM 에서 사용하는 용어다. 기존 엘리먼트는 DOM 엘리먼트를 의미한다. 리액트에서 말하는 엘리먼트는 DOM 엘리먼트의 가상표현이라고 볼 수 있다. 또한 DOM 엘리먼트에 비해 많은 정보를 담고 있기 때문에 상대적으로 크고 무겁다.

❑ 엘리먼트의 생김새

- 리액트의 엘리먼트는 자바스크립트 객체 형태로 존재한다. 또한 이 객체는 한번 생성되면 바꿀 수 없는 불변성을 가지고 있다.

```
{
        type : 'button',
    props : {
        className : 'bg-green',
        children : {
                type : 'b',
                props : {
                        children : 'Hello, element!'
                }
        }
    }
}
```

위 코드는 버튼을 나타내기 위한 엘리먼트다. 보는 것과 같이 단순한 자바스크립트 객체임을 알 수 있다.

type 에 HTML 태그 이름이 문자열로 들어가는 경우, 엘리먼트는 해당 태그 이름을 가진 DOM Node 를 나타내고 props 는 속성을 나타낸다. 위 엘리먼트가 실제로 렌더링이 된다면 아래와 같은 DOM 엘리먼트가 될 것이다.

```
<button class='bg-green'>
        <b>
        Hello, element!
        </b>
</button>
```

그렇다면 type 에 HTML 태그 이름이 문자열로 들어가는 것이 아닌 경우는 어떻게 될까?
아래 자바스크립트 코드는 리액트의 컴포넌트 엘리먼트를 나타낸 것이다. 이 역시도 일반적인 자바스크립트의 객체지만 위에 나왔던 엘리먼트와 한가지 다른 점은 type 에 HTML 태그가 아닌 컴포넌트의 이름이 들어갔다는 점이다.

```
{
        type: Button,
    props : {
        color : 'green',
        children : 'Hello, element!'
    }
}
```

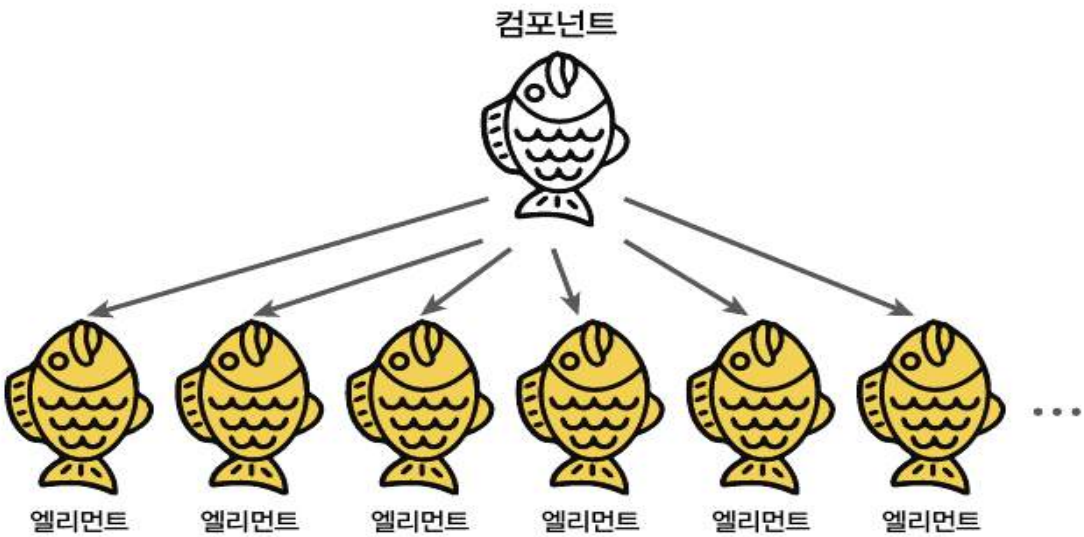
이처럼 리액트 엘리먼트는 자바스크립트 객체 형태로 존재하고 이 객체를 만드는 역할을 하는 것이 앞에서 나왔던 createElement() 함수이다. 앞에서 createElement() 함수를 호출할 때 세가지 파라미터를 넣었다.

```
React.createElement(
        type,
    [props],
    [...children]
)
```

첫 번째 파라미터로 타입이 들어가고, 두 번째는 props 가 들어간다. props 는 간단하게 엘리먼트의 속성이라고 생각하자. 그리고 세 번째로 children 이 들어가게 되는데 children 은 해당 엘리먼트의 자식 엘리먼트이다.

❑ 엘리먼트의 특징

- 리액트의 엘리먼트는 변하지 않는 성질인 불변성을 가지고 있다. 엘리먼트 생성 후에는 children 이나 attributes 를 바꿀 수 없다는 말이다.



▶ 컴포넌트와 엘리먼트

컴포넌트를 붕어빵을 만드는 틀이라고 생각하고, 엘리먼트를 붕어빵이라고 가정하자.
붕어빵 틀에 반죽을 넣고 시간이 지나 구워져 나오는 완성된 붕어빵의 속 내용을 바꿀 수 없는 것과 같은 이치라고 생각하면 이해하기 쉽다. 그렇다면 화면에 변경된 엘리먼트를 보여주기 위해서는 어떻게 해야 할까? 방법은 새로운 엘리먼트를 만들어서 기존 엘리먼트와 바꿔치기 하는 것이다.

❑ 엘리먼트 렌더링하기

- 엘리먼트를 생성한 이후에 실제로 화면에 보여주기 위해서는 렌더링이라는 과정을 거쳐야 한다.
먼저 아래 간단한 HTML 코드를 작성해보자.

```
<div id='root'></div>
```

단순해 보이지만 이 코드는 모든 리액트 앱에 필수적으로 (index.html 에) 들어가는 아주 중요한 코드이다.
실제로 이 태그 안에 리액트 엘이먼트들이 렌더링 되며 이것을 Root DOM node 라고 부른다.

이 Root DOM node 를 렌더링 하기 위해서는 아래와 같은 코드를 사용한다.

```
const element = <h1>안녕, 리액트!</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

이 코드는 먼저 엘리먼트를 하나 생성하고 생성된 엘리먼트를 root div 에 렌더링하는 코드이다. 렌더링을 위해 ReactDOM 의 render() 라는 함수를 사용하게 된다. 이 함수는 첫 번째 파라미터인 리액트 엘리먼트를 두 번째 파라미터인 HTML 엘리먼트에 렌더링 하는 역할을 한다.

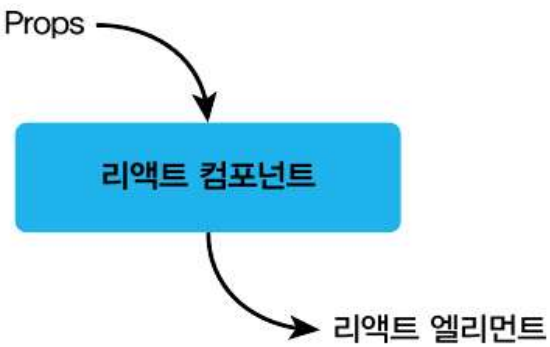
STEP 05

컴포넌트

리액트를 사용하여 애플리케이션의 인터페이스를 설계할 때 사용자가 볼 수 있는 요소는 여러 가지 컴포넌트로 구성되어 있다. 컴포넌트의 기능은 단순한 템플릿 이상이다. 데이터가 주어졌을 때 이에 맞추어 UI 를 만들어 주는 것은 물론이고 임의 메서드를 만들어 특별한 기능을 붙여 줄 수 있다.

❑ 컴포넌트 정의

- 리액트의 컴포넌트는 자바스크립트의 함수와 비슷하다. 함수가 입력을 받아서 출력을 하는 것처럼 리액트 컴포넌트도 입력을 받아서 정해진 출력을 내뱉는다.



▶ 리액트 컴포넌트

위 그림처럼 리액트 컴포넌트에서의 입력은 뒤에서 배우게 될 props 이고 출력은 앞에서 배운 리액트 엘리먼트가 된다. 결국 리액트 컴포넌트가 해주는 역할은 어떠한 속성들을 입력으로 받아서 그에 맞는 리액트 엘리먼트를 생성하여 리턴해주는 것이다. 리액트 컴포넌트는 만들고자 하는 대로 props(속성)를 넣으면 해당 속성에 맞춰 화면에 나타날 엘리먼트를 만들어 주는 것이다.

❑ 컴포넌트의 종류

- 리액트의 초기 버전에서는 클래스 컴포넌트를 주로 사용했다. 하지만 클래스 컴포넌트가 사용하기 불편하다는 의견이 많이 나왔고, 이후 함수 컴포넌트를 개선해서 주로 사용하게 됐다. 함수 컴포넌트를 개선하는 과정에서 개발 된 것이 바로 훅(Hook)이라는 것인데 이것은 뒤에서 배울 예정이다.

❑ 함수 컴포넌트

- 리액트 함수 컴포넌트의 간단한 예제를 보자.

```
function Welcome(props){
        return <h1>안녕, {props.name}</h1>;
}
```

이 코드에는 Welcome 이라는 이름을 가진 함수가 하나 나온다. 이 함수의 경우 props 의 객체를 받아서 인사말이 담긴 리액트 엘리먼트를 리턴하기 때문에 리액트 컴포넌트라고 할 수 있다.
이와 같이 생긴 것을 함수 컴포넌트라고 부른다. 함수 컴포넌트의 장점으로는 간단한 코드를 말할 수 있다.

❑ 클래스 컴포넌트

- 클래스 컴포넌트는 ES6 의 클래스를 사용해서 만들어진 형태의 컴포넌트다.

```
class Welcome extends React.Component {
        render(){
        return <h1>안녕, {this.props.name}</h1>
    }
}
```

이 코드는 위에서 살펴본 함수 컴포넌트 Welcome 과 동일한 역할을 하는 컴포넌트를 클래스 형태로 만든 것이다. 함수 컴포넌트와의 가장 큰 차이점은 리액트의 모든 클래스 컴포넌트는 React.Component 를 상속받아서 만든다는 것이다.

❑ 컴포넌트 이름 짓기

- 컴포넌트의 이름은 항상 대문자로 시작해야 된다. 이유는 소문자로 시작하는 컴포넌트를 DOM 태그로 인식하기 때문이다. 예를 들어 나 과 같이 사용하는 것은 내장 컴포넌트라는 것을 뜻하며 'div'나 'span'과 같은 문자열 형태로 React.createElement()에 전달 된다. 하지만 <Foo />와 같이 대문자로 시작하는 경우에는 React.createElement(Foo)의 형태로 컴파일 되며 자바스크립트 파일 내에서 사용자가 정의했거나 임포트한 컴포넌트를 가리킨다. 따라서 컴포넌트 이름은 항상 대문자로 시작.

```
// HTML div 태그로 인식
const element = <div />

// Welcome이라는 리액트 컴포넌트로 인식
const element = <Welcome name="인철" />;
```

위 예제 코드를 보면 첫 번째 코드는 DOM 태그를 사용하여 엘리먼트를 만드는 것이고, 두 번째 코드는 사용자가 정의한 Welcome 이라는 컴포넌트를 사용한 엘리먼트다. 만약 여기서 컴포넌트 이름이 소문자로 시작하여 welcome 이 됐다면, 리액트는 내부적으로 이것을 컴포넌트가 아니라 DOM 태그라고 인식한다. 결과적으로 에러가 발생하거나 우리가 원하는 대로 결과가 나오지 않는다.

❑ 컴포넌트 렌더링

- 컴포넌트를 만든 이후에 실제로 렌더링은 어떻게 할까? 앞에서 배운 것처럼 컴포넌트는 붕어빵 틀의 역할을 한다. 그렇기 때문에 실제로 컴포넌트가 화면에 렌더링 되는 것은 아니다. 컴포넌트라는 붕어빵 틀을 통해 찍어져서 나온 엘리먼트라는 붕어빵이 실제로 화면에 보이게 되는 것이다. 그렇다면 렌더링을 위해서는 가장 먼저 컴포넌트로부터 엘리먼트를 만들어야 한다.

```
// DOM 태그를 사용한 element
const element = <div />

// 사용자가 정의한 컴포넌트를 사용한 element
const element = <Welcome name="인철" />;
```

위 코드는 앞에 나왔던 코드와 동일한 코드인데 주석 부분의 설명이 바뀐 것이다. 두 줄의 코드 모두 리액트 엘리먼트를 만들어내게 된다. 그러면 이제 이 엘리먼트를 렌더링 하는 것이다.
실제 렌더링 하는 코드를 작성해보자.

```
function Welcome(props){
        return <h1>안녕, {props.name}</h1>
}

const element = <Welcome name="인철" />;
ReactDOM.render(
        element,
    document.getElementById('root');
);
```

위 코드에서는 먼저 Welcome 이라는 함수 컴포넌트를 선언하고 있다.
그리고 <Welcome name="인철" />라는 값을 가진 엘리먼트를 파라미터로 해서 ReactDOM.render() 함수를 호출한다. 이렇게 하면 리액트는 Welcome 컴포넌트에 { name : "인철" } 라는 props 를 넣어서 호출하고 그 결과로 리액트 엘리먼트가 생성된다. 이렇게 생성된 엘리먼트는 최종적으로 React DOM 을 통해 DOM 에 효과적으로 업데이트 되고 우리가 브라우저를 통해서 볼 수 있게 된다.

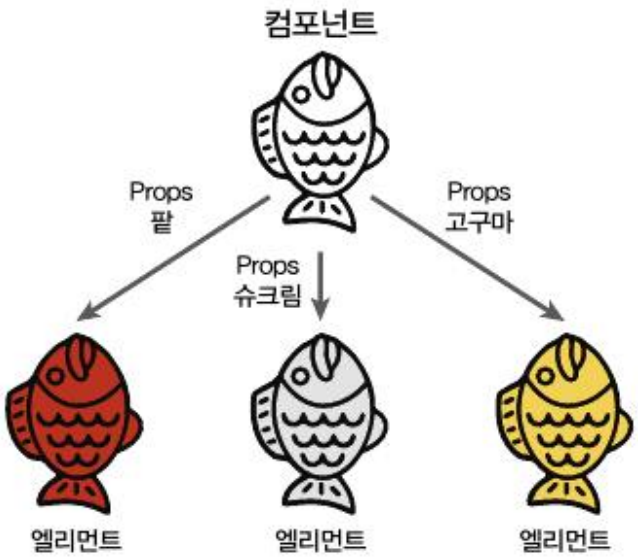
STEP 06

Props

Props 는 Properties 를 줄인 말로 컴포넌트 속성을 설정할 때 사용하는 요소이다. 컴포넌트와 마찬가지로 리액트에서 굉장히 중요한 부분이기 때문에 완벽하게 이해하고 넘어가야 한다.

❑ Props 의 개념

- props 는 prop 뒤에 복수형을 나타내는 s 를 붙여서 prop 이 여러 개인 것을 의미한다. prop 은 property 를 줄여서 쓴 것으로 속성이나 특성을 나타낸다. 여기서 속성은 리액트 컴포넌트의 속성이다. 앞서 컴포넌트는 붕어빵 틀에 비유하고 엘리먼트는 완성된 붕어빵으로 설명을 했다. 그렇다면 props 는 어떤 것일까? props 는 붕어빵에 들어가는 재료라고 생각하면 된다. 팥을 넣으면 팥붕어빵, 슈크림을 넣으면 슈크림 붕어빵, 고구마를 넣으면 고구마 붕어빵이 되는 것처럼 같은 컴포넌트에서 props 에 따라 엘리먼트의 결과가 달라지는 것을 확인할 수 있다.



▶ 컴포넌트와 Props

❑ Props 사용법

- props 는 컴포넌트에 전달할 다양한 정보를 담고 있는 자바스크립트 객체다. 컴포넌트에 props 라는 객체를 전달하기 위해 어떻게 해야 하는지 살펴보도록 한다. 먼저 JSX 를 사용하는 경우에는 아래 코드와 같이 키(key)와 값(value)로 이루어진 킷값 쌍의 형태로 컴포넌트에 props 를 넣을 수 있다.

```
function App(props){
        return(
        <Profile
                name = "황인철"
            introduction = "안녕하세요, 황인철입니다."
            age = {30}
                />
    );
}
```

이 코드에 App 컴포넌트가 나오고, 그 안에 Profile 컴포넌트를 사용하고 있다. 여기서 Profile 컴포넌트에 name, introduction, age 라는 세 가지 속성을 넣어주었다. 이렇게 하면 이 속성의 값이 모두 Profile 컴포넌트에 props 로 전달되며 props 는 아래와 같은 형태의 자바스크립트 객체가 된다.

```
{
    name : "황인철",
    introduction : "안녕하세요, 황인철입니다.",
    age : 30
}
```

name 과 introduction 에 들어간 문자열은 중괄호를 ㅈ사용하지 않았고, age 에 들어간 정수는 중괄호를 사용했다. 중괄호를 사용하면 무조건 자바스크립트 코드가 들어간다고 배웠다. 마찬가지로 props 에 값을 넣을 때에도 문자열 외에 정수, 변수, 다른 컴포넌트 등이 들어갈 경우 중괄호로 감싸주어야 한다. JSX 를 사용하지 않을 경우는 createElement() 함수를 사용해서 아래와 같이 작성한다.

```
React.createElement(
        Profile,
    {
        name : "황인철",
        introduction : "안녕하세요, 황인철입니다.",
        age : 30
    },
    null
);
```

STEP 07

State와 Lifecycle

이 장은 리액트의 컴포넌트 종류 중 주로 클래스 컴포넌트와 관련된 내용이다. 물론 state 라는 개념은 함수 컴포넌트에서도 사용하기 때문에 개념을 확실히 이해하고 넘어가는 것이 좋다. 생명주기의 경우 최근에는 클래스 컴포넌트를 거의 사용하지 않기 때문에 이런 개념이 있다 정도로 이해하고 넘어가면 된다. 특히 state 는 리액트의 핵심중의 핵심이라고 할 수 있기 때문에 반복적으로 학습해야 한다.

❑ State 란

- state 는 상태라는 뜻을 가지고 있다. 리액트에서 state 는 리액트 컴포넌트의 상태를 의미한다. 다만 상태라는 단어가 정상인지 비정상인지를 나타내는 것이라기 보다 리액트 컴포넌트의 데이터라는 의미에 가깝다. 쉽게 말해 리액트 컴포넌트의 변경 가능한 데이터를 state 라고 부른다.
  state 를 정의할 때 중요한 점은 꼭 렌더링이나 데이터 흐름에 사용되는 값만 state 에 포함시켜야 한다는 것이다. state 가 변경될 경우 컴포넌트가 재렌더링 되기 때문에 렌더링과 데이터 흐름에 관련 없는 값을 포함하면 컴포넌트가 다시 렌더링되어 성능을 저하 시킬 수 있다. 따라서 렌더링과 데이터 흐름에 관련된 값만 state 에 포함하도록 해야 하며, 그렇지 않은 값은 컴포넌트 인스턴스의 필드로 정의하면 된다.

❑ State 의 특징

- state 는 따로 복잡한 형태가 있는 것이 아니라 하나의 자바스크립트 객체다.

```
class LikeButton extends React.Component {
        constructor(props){
        super(props);
        this.state = {
                liked : false
        };
    }
    .....
}
```

이 코드는 LikeButton 이라는 리액트 클래스 컴포넌트를 나타낸 것이다. 모든 클래스 컴포넌트에는 constructor 라는 이름의 함수가 존재하는데 우리말로 생성자라는 의미를 갖고 있으며, 클래스가 생성될 때 실행되는 함수이다. 이 생성자 코드를 보면 this.state 라는 부분이 나오는데 이 부분이 바로 현재 컴포넌트의 state 를 정희하는 부분이다. 클래스 컴포넌트의 경우 state 를 생성자에서 정의한다. 함수 컴포넌트는 state 를 useState()라는 훅을 사용해서 정의하게 되는데 이 부분은 뒤에서 자세히 다루도록 한다. 이렇게 정의한 state 는 정의된 이후 일반적인 자바스크립트 변수를 다루듯이 직접 수정할 수 없다. 엄밀히 말하면 수정이 가능하긴 하지만 그렇게 하면 안된다.

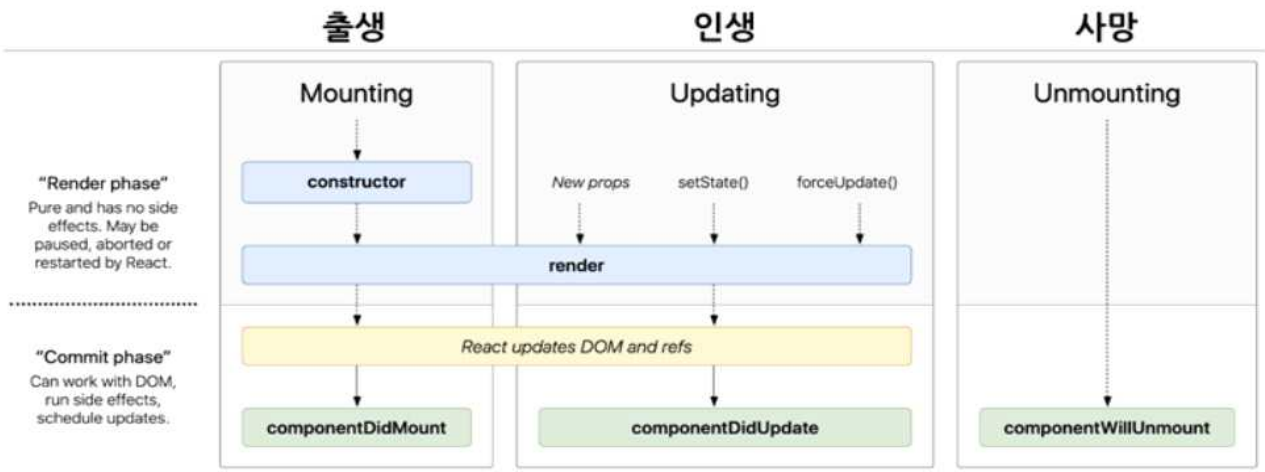
```
// state를 직접 수정(잘못된 사용법)
this.state = {
        name : 'icia'
};

// setState 함수를 통한 수정(정상적인 방법)
this.setState({
        name : 'icia'
});
```

첫 번째 방법은 state 를 직접 수정하는 방법이고, 두 번째 방법은 setState() 함수를 통해 수정하는 방법이다. state 를 직접 수정할 수는 없지만 리액트가 수정된 것을 제대로 인지하지 못할 수 있기 때문에 애초에 state 는 직접적인 변경이 불가능하다고 생각해야 한다. 그래서 state 를 변경하고자 할 때는 반드시 setState() 함수를 사용해야 한다.

❑ 생명주기 Lifecycle

- 리액트 컴포넌트가 생성되는 시점과 사라지는 시점이 존재한다. 아래 그림은 컴포넌트의 생명주기이다.



▶ 컴포넌트의 생명주기

생명주기를 보면 출생, 인생, 사망으로 나누어져 있다. 각 과정의 하단에 초록색으로 표시된 부분은 생명주기에 따라 호출되는 클래스 컴포넌트 함수이다. 이 함수들을 Lifecycle method 라고 부르며 생명주기 함수가 된다. 먼저 컴포넌트가 생성되는 시점, 사람으로 말하는 출생을 살펴보자. 이 과정을 마운트(Mount)라고 부르는데 이때 컴포넌트의 생성자(constructor)가 실행된다. 앞에서 본 것처럼 생성자에서는 컴포넌트의 state 를 정의하게 된다. 또한 컴포넌트가 렌더링되며 이후에 componentDidMount() 함수가 호출된다. 이후에 리액트 컴포넌트도 생애 동안 변화를 겪으면서 여러번 렌더링 된다. 이를 리액트 컴포넌트로 말하면 업데이트(Update)되는 과정이라고 할 수 있다. 업데이트 과정에서는 컴포넌트의 props 가 변경되거나 setState() 함수 호출에 의해 state 가 변경되거나, forceUpdate()라는 강제 업데이트 함수 호출로 인해 컴포넌트가 다시 렌더링 된다. 그리고 렌더링 이후에 componentDidUpdate() 함수가 호출된다. 마지막으로 리액트 컴포넌트가 사라지는 과정을 겪는데 이를 언마운트(Unmount)라고 부른다. 상위 컴포넌트에서 현재 컴포넌트를 더 이상 화면에 표시하지 않게 될 때 언마운트 된다고 볼 수 있다. 이 때 언마운트 직전에 componentWillUnmount() 함수가 호출된다. 컴포넌트 생명주기에서 기억해야 할 부분은 컴포넌트가 계속 존재하는 것이 아니라 흐름에 따라 생성되고 업데이트 되다가 사라진다는 점이다.

STEP 08

Hooks

훅(hook)은 리액트가 처음 나왔을 때부터 있던 개념은 아니고, 리액트 버전 16.8 에서 새롭게 등장한 개념이다. 최근에는 리액트로 개발할 때 함수 컴포넌트와 대부분 훅을 사용하기 때문에 훅에 대해 잘 이해해야 한다.

❑ 훅(Hook)이란?

- 클래스 컴포넌트에서는 생성자에서 state 를 정의하고 setState() 함수를 통해 state 를 업데이트 한다. 이처럼 클래스 컴포넌트는 state 와 관련된 기능뿐만 아니라 컴포넌트의 생명주기 함수들까지 모두 명확하게 정의되어 있기 때문에 잘 가져다 쓰기만 하면 된다. 하지만 기존 함수 컴포넌트는 클래스 컴포넌트와는 다르게 코드도 굉장히 간결하고, 별도로 state 를 정의해서 사용하거나 컴포넌트의 생명주기에 맞춰 어떤 코드가 실행되도록 할 수 없었다. 따라서 함수 컴포넌트에 이런 기능을 지원하기 위해서 나온 것이 바로 훅이다. 훅을 사용하면 함수 컴포넌트도 클래스 컴포넌트의 기능을 모두 동일하게 구현할 수 있게 되는 것이다.

- 리액트 훅은 리액트의 state 와 생명주기 기능을 원하는 시점에 정해진 함수를 실행되도록 만든 것이다. 이때 실행되는 함수를 훅이라고 부르기로 한 것이다. 이러한 훅의 이름은 모두 use 로 시작한다. 훅이 수행하는 기능에 따라서 이름을 짓게 되었는데 각 기능을 사용하겠다는 의미로 use 를 앞에 붙였다. 개발자가 직접 커스텀 훅(Custom Hook)을 만들어서 사용할 수 있는데 커스텀 훅은 개발자 마음대로 이름을 지을 수 있지만 뒤에서 배울 훅의 규칙에 따라 이름 앞에 use 를 붙여서 훅이라는 것을 나타내야 한다.

❑ useState

- 가장 대표적이고 많이 사용하는 훅으로 useState()가 있다. useState()는 이름에서 알 수 있듯이 state 를 사용하기 위한 훅이다. 함수 컴포넌트에서는 기본적으로 state 라는 것을 제공하지 않기 때문에 클래스 컴포넌트처럼 state 를 사용하고 싶으면 useState() 훅을 사용해야 한다. 아래는 간단한 예제 코드이다.

```
import React, { useState } from "react";

function Counter(props) {
        var count = 0;

    return(
        <div>
                <p>총 {count}번 클릭했습니다.</p>
            <button onClick={() => count++}>
                클릭
                        </button>
                </div>
    );
}
```

위 코드에는 Counter 라는 함수 컴포넌트가 등장한다. Counter 컴포넌트는 버튼을 클릭하면 카운트를 하나씩 증가시키고 현재 카운트를 보여주는 단순한 컴포넌트다. 그런데 만약 위처럼 카운트를 함수의 변수로 선언해서 사용하게 되면 버튼 클릭 시 카운트 값을 증가시킬 수는 있지만, 재렌더링(Re-rendering)이 일어나지 않아 새로운 카운트 값이 화면에 표시되지 않게 된다. 이런 경우 state 를 사용해서 값이 바뀔 때마다 재렌더링이 되도록 해야 하는데, 함수 컴포넌트에는 해당 기능이 따로 없기 때문에 useState()를 사용하여 state 를 선언하고 업데이트해야 한다. useState()는 아래와 같이 사용한다.

```
const [변수명, set함수명] = useState(초깃값);
```

useState()를 호출할 때에는 파라미터로 선언할 state 의 초깃값이 들어간다. 클래스 컴포넌트의 생성자에서 state 를 선언할 때 초깃값을 넣어 주는 것과 동일한 것이라고 보면 된다. 이렇게 초깃값을 넣어 useState()를 호출하면 리턴 값으로 배열이 나온다. 리턴된 배열에는 두가지 항목이 들어있는데 첫 번째 항목은 state 로 선언된 변수이고 두 번째 항목은 해당 state 의 set 함수다.

useState()를 사용하는 코드의 예를 보도록 한다.

```
import React, { useState } from "react";

function Counter(props) {
        const [count, setCount] = useState(0);

    return (
        <div>
                <p>총 {count}번 클릭했습니다.</p>
            <button onClick={() => setCount(count+1)}>
                클릭
            </button>
        </div>
    );
}
```

위의 코드는 useState()를 사용하여 카운트 값을 state 로 관리하도록 만든 것이다. 이 코드에서 state 의 변수명과 set 함수가 각각 count, setCount 로 되어있는 것을 볼 수 있다. 버튼이 눌렸을 때 setCount() 함수를 호출해서 카운트를 1 증가시킨다. 그리고 count 의 값이 변경되면 컴포넌트가 재렌더링 되면서 화면에 새로운 카운트 값이 표시된다. 이 과정은 클래스 컴포넌트에서 setState() 함수를 호출해서 state 가 업데이트되고 이후 컴포넌트가 재렌더링되는 과정과 동일하다고 보면 된다. 다만 클래스 컴포넌트에서는 setState() 함수 하나를 사용해서 모든 state 값을 업데이트할 수 있었지만 useState()를 사용하는 방법에서는 변수 각각에 대해 set 함수가 따로 존재한다는 것을 기억해야 한다.

❑ useEffect

- useState()와 같이 가장 많이 사용되는 훅으로 useEffect()가 있다. useEffect()는 사이드 이펙트(side effect)를 수행하기 위한 훅이다. 사이드 이펙트는 사전적인 의미로 부작용으로 쓰이지만 리액트에서 말하는 사이드 이펙트는 그냥 효과 혹은 영향을 뜻하는 이펙트의 의미에 가깝다. 예를 들면 서버에서 데이터를 받아오거나 수동으로 DOM 을 변경하는 등의 작업을 의미한다. 이런 작업을 이펙트라고 부르는 이유는 이 작업들이 다른 컴포넌트에 영향을 미칠 수 있으며 렌더링 중에는 작업이 완료될 수 없기 때문이다. 렌더링이 끝난 이후에 실행되어야 하는 작업들이다. useEffect()는 리액트의 함수 컴포넌트에서 사이드 이펙트를 실행할 수 있도록 해주는 훅이다. useEffect()는 클래스 컴포넌트에서 제공하는 생명주기 함수인 componentDidMount(), componentDidUpdate(), componentWillUnmount()와 동일한 기능을 하나로 통합해서 제공한다. 그래서 useEffect() 훅만으로 위의 생명주기 함수와 동일한 기능을 수행할 수 있다. useEffect()는 아래와 같이 사용한다.

```
useEffect(이펙트 함수, 의존성 배열);
```

첫 번째 파라미터로는 이펙트 함수가 들어가고, 두 번째 파라미터로 의존성 배열이 들어간다. 의존성 배열은 말 그대로 이 이펙트가 의존하고 있는 배열인데 배열 안에 있는 변수 중에 하나라도 값이 변경되었을 때 이펙트 함수가 실행된다. 기본적으로 이펙트 함수는 처음 컴포넌트가 렌더링된 이후와 업데이트로 인한 재렌더링 이후에 실행된다. 만약 이펙트 함수가 마운트와 언마운트시에 단 한 번씩만 실행되게 하고 싶으면, 의존성 배열에 빈 배열([ ])을 넣으면 된다. 이렇게 하면 해당 이펙트가 props 나 state 에 어떤 값에도 의존하지 않는 것이 되므로 여러번 실행되지 않는다. 의존성 배열은 생략할 수도 있는데 생략하게 되면 컴포넌트가 업데이트될 때마다 호출이 된다. 아래 useEffect()를 사용한 코드를 보자.

```
import React, { useState, useEffect } from "react";

function Counter(props) {
        const [count, setCount] = useState(0);

    // componentDidMount, componentDidUpdate와 비슷하게 작동한다.
    useEffect(() => {
        // 브라우저 API를 사용해서 document의 title을 업데이트 합니다.
        document.title = `총 ${count}번 클릭했습니다.`;
    });

    return (
        <div>
                <p>총 {count}번 클릭했습니다.</p>
            <button onClick={() => setCount(count+1)}>
                클릭
            </button>
        </div>
    );
}
```

위의 코드는 앞에서 useState()를 배울 때 살펴 본 코드와 거의 동일하며 추가로 useEffect()훅을 사용하고 있다. useEffect()를 사용해서 클래스 컴포넌트에서 제공하는 componentDidMount(), componentDidUpdate()와 같은 생명주기 함수의 기능을 동일하게 수행하도록 만들었다. useEffect() 안에 있는 이펙트 함수에서는 브라우저에서 제공하는 API 를 사용해서 document 의 title 을 업데이트 한다. document 의 title 은 브라우저에서 페이지를 열었을 때 창에 표시되는 문자열이다. 크롬 부라우저의 경우 탭에 나오는 제목이라고 보면 된다. 위 코드처럼 의존성 배열 없이 useEffect()를 사용하면 리액트는 DOM 이 변경된 이후에 해당 이펙트 함수를 실행하라는 의미로 받아들인다. 그래서 기본적으로 컴포넌트가 처음 렌더링 될 때를 포함해서 매번 렌더링될 때마다 이펙트가 실행된다고 보면 된다. 또한 이펙트 함수 컴포넌트 안에서 선언되기 때문에 해당 컴포넌트의 props 와 state 에 접근할 수도 있다. 위 코드에서는 count 라는 state 에 접근하여 해당 값이 포함된 문자열을 생성해서 사용하는 것을 볼 수 있다. 그렇다면 componenWillUnmount()와 동일한 기능은 useEffect()로 어떻게 구현할까?

```
import React, { useState, useEffect } from "react";

function UserStatus(props) {
        const [isOnline, setIsOnline] = useState(null);

    function handleStatusChange(status) {
        setIsOnline(status.isOnline);
    }

    useEffect(() => {
        ServerAPI.subscribeUserStatus(props.user.id, handleStatusChange);
        return () => {
                ServerAPI.unsubscribeUserStatus(props.user.id, handleStatusChange);
        };
    });

    if(isOnline === null) {
        return '대기 중...';
    }
    return isOnline ? '온라인' : '오프라인';
}
```

위 코드는 useEffect()에서 먼저 ServerAPI 를 사용하여 사용자의 상태를 구독하고 있다. 이후 함수를 하나 리턴하는데 해당 함수 안에는 구독을 해지하는 API 를 호출하도록 되어 있다. useEffect()에서 리턴하는 함수는 컴포넌트가 마운트 해제될 때 호출된다. 결과적으로 useEffect()의 리턴 함수의 역할은 componentWillUnmount()함수가 하는 역할과 동일하다. 또한 userEffect() 훅을 하나의 컴포넌트에서 여러개 사용할 수도 있다.

```
useEffect(() => {
    // 컴포넌트가 마운트 된 이후,
    // 의존성 배열에 있는 변수들 중 하나라도 값이 변경되었을 때 실행됨
    // 의존성 배열에 빈 배열([])을 넣으면 마운트와 언마운트시에 단 한 번씩만 실행됨
    // 의존성 배열 생략 시 컴포넌트 업데이트 시마다 실행됨
    ...

    return () => {
        // 컴포넌트가 마운트 해제되기 전에 실행됨
        ...
    }
}, [의존성 변수1, 의존성 변수2, ...]);
```

❑ useMemo

- useMemo() 훅은 Memoized value를 리턴하는 훅이다. 파라미터로 Memoized value를 생성하는 create 함수와 의존성 배열을 받습니다. 뒤에 나올 메모이제이션의 개념처럼 의존성 배열에 들어있는 변수가 변했을 경우에만 새로 create 함수를 호출하여 결괏값을 반환하며, 그렇지 않을 경우에는 기존 함수의 결괏값을 그대로 반환한다. useMemo() 훅을 사용하면 컴포넌트가 다시 렌더링될 때마다 연산량이 높은 작업을 반복하는 것을 피할 수 있다. 결과적으로 빠른 렌더링 속도를 얻을 수 있다.

```
const memoizedValue = useMemo(
    () => {
        // 연산량이 높은 작업을 수행하여 결과를 반환
        return computeExpensiveValue(의존성 변수1, 의존성 변수2);
    },
    [의존성 변수1, 의존성 변수2]
);
```

useMemo() 훅을 사용할ㄹ 때 기억해야 할 점은 useMemo()로 전달된 함수는 렌더링이 일아나는 동안 실행된다는 점이다. 서버에서 데이터를 받아오거나 수동으로 DOM을 변경하는 작업 등은 렌더링이 일어나는 동안 실행돼서는 안되기 때문에 useMemo() 훅의 함수에 넣으면 안되고 useEffect()훅을 사용해야 한다.

❑ useCallback

- useCallback() 훅은 이전에 나온 useMemo() 훅과 유사한 역할을 한다. 한 가지 차이점은 값이 아닌 함수를 반환한다는 점이다. useCallback() 훅은 useMemo() 훅과 마찬가지로 함수와 의존성 배열을 파라미터로 받는다. useCallback() 훅에서는 파라미터로 받는 이 함수를 콜백이라고 부른다. 그리고 의존성 배열에 있는 변수 중 하나라도 변경되면 Momoized(메모이제이션이 된) 콜백 함수를 반환한다.

```
const memoizedCallback = useCallback(
    () => {
        doSomething(의존성 변수1, 의존성 변수2);
    },
    [의존성 변수1, 의존성 변수2]
);
```

의존성 배열에 따라 Memoized 값을 반환한다는 점에서는 useMemo() 훅과 완전히 동일하다. 그래서 useCallback(function, dependencies)은 useMemo(() => function, dependencies)와 동일하다고 볼 수 있다.

❑ useRef

- useRef() 훅은 레퍼런스를 사용하기 위한 훅이다. 리액트에서 레퍼런스란 특정 컴포넌트에 접근할 수 있는 객체를 의미한다. useRef() 훅은 바로 레퍼런스 객체를 반환한다. 레퍼런스 객체에는 .current 라는 속성이 있는데 이것은 현재 레퍼런스 하고 있는 엘리먼트를 의미한다고 보면 된다.

```
const refContainer = useRef(초깃값);
```

위와 같이 useRef() 훅을 사용하면 파라미터로 들어온 초깃값으로 초기화된 레퍼런스 객체를 반환한다. 만약 초깃값이 null 이라면 .current 의 값이 null 인 레퍼런스 객체가 반환되고 이렇게 반환된 레퍼런스 객체는 컴포넌트의 라이프타임 전체에 걸쳐서 유지된다. 즉, 컴포넌트 마운트 해제 전까지는 계속 유지된다는 것이다. 쉽게 말해 useRef()훅은 변경 가능한 .current 라는 속성을 가진 하나의 상자라고 생각하면 된다.

```
function TextInputWithFocusButton(props) {
        const inputElem = useRef(null);

    const onButtonClick = () => {
        // `current`는 마운트 된 input element를 가리킴
        inputElem.current.focus();
    };

    return (
        <>
                <input ref={inputElem} type="text" />
            <button onClick={onButtonClick}>Focus the input</button>
        </>
    );
}
```

위의 코드는 useRef() 훅을 사용하여 버튼 클릭 시 <input>에 포커스를 하도록 하는 코드다. 초깃값으로 null 을 넣었고 결과로 반환된 inputElem 이라는 레퍼런스 객체를 <input>태그에 넣어줬다. 그리고 버튼 클릭 시 호출되는 함수에서 inputElem.current 를 통해 실제 엘리먼트에 접근하여 focus() 함수를 호출한다.

❑ hook 의 규칙

- 훅은 단순한 자바스크립트 함수이지만 두 가지 지켜야 할 규칙이 있다. 첫 번째 규칙은 훅은 무조건 최상위 레벨에서만 호출해야 한다는 것이다. 여기서 말하는 최상위 레벨은 리액트 함수 컴포넌트의 최상위 레벨을 의미한다. 따라서 반복문이나 조건문 또는 중첩된 함수들 안에서 훅을 호출하면 안 된다는 뜻이다. 이 규칙에 따라서 훅은 컴포넌트가 렌더링 될 때마다 매번 같은 순서로 호출되어야 한다. 그렇게 해야 리액트가 다수의 useState() 훅과 useEffect() 훅의 호출에서 컴포넌트의 state 를 올바르게 관리할 수 있게 된다.

```
function MyComponent(props) {
        const [name, setName] = useState('icia');

    if(name !== '') {
        useEffect( () => {
                ...
        });
    }
    .....
}
```

위 코드에서 name !== ' ' 라는 조건문의 값이 참인 경우에만 useEffect() 훅을 호출 하도록 되어 있다. 이런 경우 중간에 name 의 값이 빈 문자열이 되면 useEffect() 훅이 호출되지 않는다. 결과적으로 렌더링할 때마다 훅이 같은 순서대로 호출되는 것이 아니라 조건문의 결과에 따라 호출되는 훅이 달라지므로 잘못된 코드다. 훅은 꼭 최상위 레벨에서만 호출해야 한다는 점을 기억하자.
두 번째 규칙은 리액트 함수 컴포넌트에서만 훅을 호출해야 한다는 것이다. 그렇기 때문에 일반적인 자바스크립트 함수에서 훅을 호출하면 안 된다. 훅은 리액트 함수 컴포넌트에서 호출하거나 직접 만든 커스텀 훅에서만 호출할 수 있다. 이 규칙에 따라 리액트 컴포넌트에 있는 state 와 관련된 모든 로직은 소스코드를 통해 명확하게 확인이 가능해야 한다.

STEP 09

이벤트 핸들링

이 장에서는 이벤트를 다루는 방법에 대해 배워보도록 하겠다. 흔히 이벤트라고 하면 연인에게 해주는 깜짝파티 같은 것을 떠올릴 수도 있을텐데 컴퓨터 프로그래밍에서의 이벤트는 사건 이라는 의미를 가지고 있다. 예를 들면 사용자가 버튼을 클릭하는 사건도 하나의 이벤트라고 볼 수 있다. 여기서 클릭한 사건을 클릭 이벤트라고 부른다. 웹사이트에는 굉장히 다양한 이벤트들이 있다. 이러한 이벤트들이 발생했을 때 원하는 대로 처리를 잘 해줘야 웹사이트가 정상적으로 돌아간다.

❑ 이벤트 처리하기

- 아래는 DOM 에서 클릭 이벤트를 처리하는 예제 코드다. 버튼이 눌리면 activate() 라는 함수를 호출하도록 만들었다. DOM 에서는 클릭 이벤트를 처리할 함수를 onclick 을 통해서 전달한다.

```
<button onclick="activate()">
        Activate
</button>
```

리액트에서는 이벤트를 처리할 때 조금 다른 부분이 있다. 먼저 이벤트 이름인 onclick 이 onClick 으로 카멜표기법이 적용된다. 또 다른 점은 DOM 에서는 이벤트를 처리할 함수를 문자열로 전달하지만 리액트는 함수 그대로 전달한다.

```
<button onClick={activate}>
        Activate
</button>
```

이처럼 DOM 에도 이벤트가 있고 리액트에도 이벤트가 있지만 사용하는 방법이 조금 다르다는 것을 기억한다. 리액트에서 해당 이벤트를 처리하는 함수를 이벤트 핸들러 라고 부른다. 또는 이벤트가 발생하는 것을 계속 듣고 있다는 의미로 이벤트 리스너 라고 부르기도 한다.

그렇다면 이벤트 핸들러는 어떻게 추가해야 할까?

```
class Toggle extends React.Component {
        constructor(props){
        super(props);

        this.state = { isToggleOn: true };

        // callback에서 `this`를 사용하기 위해서는 바인딩을 필수적으로 해줘야 한다.
        this.handleClick = this.handleClick.bind(this);
    }

    handleClick() {
        this.setState(prevState => ({
                isToggleOn: !prevState.isToggleOn
        }));
    }

    render(){
        return(
                <button onClick={this.handleClick}>
                {this.state.isToggleOn ? '켜짐' : '꺼짐'}
            </button>
        );
    }
}
```

Toggle 이라는 클래스 컴포넌트의 state 에는 isToggleOn 이라는 Boolean 변수가 하나가 있다. 버튼을 클릭하면 이벤트 핸들러 함수인 handleClick() 함수를 호출하도록 되어 있는데, 여기서 눈여겨봐야 할 곳은 바로 handleClick() 함수를 정의하는 부분과 this.handleClick = this.handleClick.bind(this); 부분이다.
먼저 handleClick() 함수의 정의 부분은 일반적인 함수를 정의하는 것과 동일하게 괄호와 중괄호를 사용해서 클래스 함수로 정의하고 있다. 이렇게 정의된 함수를 constructor()에서 bind()를 이용하여 this.handleClick 에 대입해 준다. JSX 에서 this 의 의미에 대해 유의해야 하는데 bind 를 하는 이유는 자바스크립트에서 기본적으로 클래스 함수들이 바운드 되지 않기 때문에 bind 를 하지 않으면 this.handleClick 은 글로벌 스코프에서 호출되는데 글로벌 스코프에서 this.handleClick 은 undefined 이므로 사용할 수 없다.
이것은 리액트에만 해당되는 내용이 아니라 자바스크립트 함수의 작동 원리 중 일부분이다. 다만 지금은 클래스 컴포넌트를 거의 사용하지 않기 때문에 위 내용은 참고로만 알고 있고 함수 컴포넌트로 변환하면 아래와 같이 나타낼 수 있다.

```
function Toggle(props) {
        const [isToggleOn, setIsToggleOn] = useState(true);

    // 방법1. 함수 안에 함수로 정의
    functino handleClick() {
        setToggleOn((isToggleOn) => !isToggleOn);
    }

    // 방법2. arrow functino을 사용하여 정의
    const handleClick = () => {
        setIsToggleOn((isToggleOn) => !isToggleOn);
    }

    return(
        <button onClick={handleClick}>
                {isToggleOn ? "켜짐" : "꺼짐"}
        </button>
    );
}
```

함수 컴포넌트 내부에서 이벤트 핸들러를 정의하는 방법은 함수 안에 또 다른 함수로 정의하는 방법과 arrow function 을 사용하여 정의하는 방법이 있다. 함수 컴포넌트에서는 this 를 사용하지 않고 onClick 에 곧바로 handleClick 을 넘기면 된다.

❑ Arguments(매개변수) 전달하기

- Arguments 는 함수에 전달할 데이터이고 우리말로는 매개변수라고 부른다. 이벤트 핸들러에 매개변수를 전달해야 하는 경우가 굉장히 많다. 예를 들어 특정 사용자 프로필을 클릭했을 때 해당 사용자의 아이디를 매개변수로 전달해서 정해진 작업을 처리해야 하는 경우를 들 수 있다. 아래 함수 컴포넌트의 예제 코드를 통해 어떤 식으로 매개변수를 이벤트 핸들러에 전달하는지 보도록 한다.

```
function MyButton(props) {
        const handleDelete = (id, event) => {
        console.log(id, event.target);
    };

    return (
        <button onClick={(event) => handleDelete(1, event)}>
                삭제하기
                </button>
    );
}
```

STEP 10

조건부 렌더링

리액트에서 꼭 필요한 개념은 아니지만 자주 사용하게 되는 기능이기 때문에 사용하는 방법과 원리에 대해 잘 이해하고 넘어가도록 한다.

❑ 조건부 렌더링이란?

• 조건부 렌더링(Conditional Rendering)란 Condition(조건, 상태)에 따라 렌더링이 달라지는 것을 의미한다. 조건문을 통해 결과가 true 나 false 가 나오는데 이 결과에 따라서 렌더링을 다르게 하는 것이다.

```
function UserGreeting(props) {
        return <h1>다시 오셨군요!</h1>
}

function GuestGreeting(props) {
        return <h1>회원가입을 해주세요.</h1>
}
```

UserGreeting 과 GuestGreeting 이라는 두 개의 함수 컴포넌트가 있다. UserGreeting 컴포넌트는 이미 회원인 사용자에게 보여줄 메시지를 출력하는 컴포넌트 이고 GuestGreeting 컴포넌트는 아직 가입하지 않은 게스트 사용자에게 보여줄 메시지를 출력하는 컴포넌트다.

```
function Greeting(props) {
        const isLoggedIn = props.isLoggedIn;

    if(isLoggedIn) {
        return <UserGreeting />;
    }
    return <GuestGreeting />;
}
```

Greeting 컴포넌트는 isLoggedIn 이라는 변수의 값이 true 면 UserGreeting 컴포넌트를 리턴하고, 그렇지 않으면 GuestGreeting 컴포넌트를 리턴한다. props 로 들어오는 isLoggedIn 의 값에 따라 화면에 출력되는 결과가 달라지게 된다. 이처럼 조건에 따라 결과가 달라지는 것을 조건부 렌더링이라 한다.

❑ 엘리먼트 변수

- 조건부 렌더링을 사용하다 보면 렌더링해야 될 컴포넌트를 변수처럼 다루고 싶을 때가 있다. 이때 사용할 수 있는 방법이 바로 엘리먼트 변수이다. 엘리먼트 변수는 이름 그대로 리액트 엘리먼트를 변수처럼 다루는 방법이다.

```
function LoginButton(props) {
        return(
        <button onClick={props.onClick}>
                로그인
        </button>
    );
}

function LogoutButton(props) {
        return(
        <button onClick={props.onClick}>
                로그아웃
        </button>
    );
}
```

위 코드는 LoginButton 과 Logout 두 개의 컴포넌트가 있다. 각 컴포넌트는 이름의 의미처럼 로그인 버튼과 로그아웃 버튼을 나타낸다. 아래 나오는 LoginControl 컴포넌트에는 사용자의 로그인 여부에 따라 두 개의 컴포넌트를 선택적으로 보여주게 된다.

```
function LoginControl(props) {
        const [isLoggedIn, setIsLoggedIn] = useState(false);

    const handleLoginClick = () => {
        setIsLoggedIn(true);
    }

    const handleLogoutClick = () => {
        setIsLoggedIn(false);
    }
```

```
    let button;
    if(isLoggedIn) {
        button = <LogoutButton onClick={handleLogoutClick} />;
    } else {
        button = <LoginButton onClick={handleLoginClick} />;
    }

    return(
        <div>
                <Greeting isLoggedIn={isLoggedIn} />
            {button}
        </div>
    );
}
```

isLoggedIn 의 값에 따라서 button 이라는 변수에 컴포넌트를 대입하는 것을 볼 수 있다. 이렇게 컴포넌트가 대입된 변수를 return 에 넣어 실제로 컴포넌트가 렌더링 되도록 만들고 있다. 설명은 컴포넌트라고 했지만 실제로는 컴포넌트로부터 생성된 리액트 엘리먼트가 된다. 이처럼 엘리먼트를 변수처럼 저장해서 사용하는 방법을 엘리먼트 변수라고 부른다. 이렇게 별도로 변수를 선언해서 조건부 렌더링을 할 수도 있지만 다음에 나오는 Inline 조건문을 사용해서 더 간결하게 코드를 작성해 보도록 한다.

❑ 인라인 조건

- 인라인 조건은 조건문을 코드 안에 집어 넣는다는 뜻을 가지고 있다. 인라인 If 는 if 문을 필요한 곳에 직접 넣어서 사용하는 방법이다. 다만 실제로 if 문을 넣는 것은 아니고 if 문과 동일한 효과를 내기 위해 && 라는 논리 연산자를 사용한다. &&연산자는 흔히 AND 연산이라고 부르는데 양쪽에 나오는 조건문이 모두 true 인 경우에만 전체 결과가 true 가 된다.

```
true && expression -> expression
false && expression -> false
```

조건문이 true 뒤에 나오면 expression 이 평가되고, false 뒤에 나오면 expression 에 상관없이 결과는 false 값을 갖는다.

```
function Mailbox(props) {
        const unreadMessages = props.unreadMessages;

    return(
        <div>
                <h1>안녕하세요!</h1>
            {unreadMessages.length > 0 &&
                <h2>
                        현재 {unreadMessages.length}개의 읽지 않은 메세지가 있습니다.
                </h2>
            }
        </div>
    );
}
```

위 코드에서 unreadMessages.length > 0 의 값에 따라 뒤에 나오는 <h2>태그 부분이 렌더링 되거나 안되거나 한다. && 연산자를 사용하는 이런 패턴은 단순하지만 리액트에서 많이 사용하는 조건이기 때문에 잘 기억해둬야 한다.

- if 문의 경우 보여주거나 안보여 주거나 두 가지 경우만 있었지만 인라인 If-Else 조건은 값에 따라 다른 엘리먼트를 보여줄 때 사용한다. 흔히 삼항 연산자라고 부르는 ? 연산자를 사용한다.

```
function UserStatus(props) {
        return(
        <div>
                이 사용자는 현재 <b>{props.isLoggedIn ? '로그인 한' : '로그인 안한'}</b> 상태
        </div>
    );
}
```

isLoggedIn 의 값이 true 인 경우에는 '로그인 한' 이라는 문자열을 출력하고, false 인 경우에는 '로그인 안한' 이라는 문자열을 출력한다. 아래와 같이 꼭 문자열이 아닌 엘리먼트를 넣을 수도 있다.

```
function LgoinControl(props) {
        const[isLoggedIn, setIsLoggedIn] = useState(false);

    const handleLoginClick = () => {
        setIsLoggedIn(true);
    }

    const handleLogoutClick = () => {
        setIsLoggedIn(false);
    }

    return(
        <div>
                <Greeting isLoggedIn={isLoggedIn} />
            {isLoggedIn
                ? <LogoutButton onClick={handleLogoutClick} />
                : <LoginButton onClick={handleLoginClick} />
            }
        </div>
    );
}
```

❑ 컴포넌트 렌더링 막기

- 컴포넌트를 렌더링하고 싶지 않을 때에는 null 값을 리턴하면 된다. 리액트에서는 null 을 리턴하면 렌더링 되지 않기 때문이다.

```
function WarningBanner(props) {
    if(!props.warning) {
        return null;
    }

    return (
        <div>경고!</div>
    );
}
```

WarningBanner 라는 컴포넌트는 props.warning 의 값이 false 인 경우 null 을 리턴하고 true 인 경우에만 경고 메시지를 출력하는 컴포넌트다. WarningBanner 컴포넌트를 실제로 사용하는 코드를 확인해보자.

```
function MainPage(props) {
        const [showWarning, setShowWarning] = useState(false);

    const handleToggleClick = () => {
        setShowWarning(prevShowWarning => !prevShowWarning);
    }

    return(
        <div>
                <WarningBanner warning={showWarning} />
            <button onClick={handleToggleClick}>
                {showWarning ? '감추기' : '보이기'}
            </button>
        </div>
    )
}
```

mainPage 컴포넌트는 showWarning 이라는 state 의 값을 WarningBanner 컴포넌트의 props 로 전달하여 showWarning 의 값에 따라 경고문을 표시하거나 또는 표시하지 않게 된다. 이처럼 렌더링을 하고 싶지 않을 경우 null 을 리턴하면 된다.